Webpack

Objectifs

L'objectif de cette démonstration est la mise en place d'un projet HTML/Javascript simple en utilisant npm et Webpack.

Webpack est la petite nouveauté par rapport à ce qu'ils connaissent déjà. Pour illustrer l'intérêt de celui-ci, on va installer le framework scss tailwind.

Mise en place

- Ouvrir Webstorm
- Vérifier que nodejs est installé

```
node -v
```

- À l'intérieur d'un nouveau dossier, initialiser un projet javascript

```
npm init -y
```

Cette commande va créer un fichier package.json

- Créer un nouveau document javascript : clic droit ⇒New ⇒JavaScript file
- Ainsi qu'un nouveau document html : clic droit ⇒New ⇒HTML file

le lien entre le javascript et le html sera fait par Webpack

Webpack

Installation

Grâce à NPM, tout se fait en ligne de commande.

À l'intérieur du dossier de notre projet de démonstration :

```
npm install webpack webpack-cli --save-dev
```

--save-dev permet de dire à webpack de ne pas incorporer ces deux paquets à notre projet final.

Le fichier *package.json* a dû s'enrichir de quelques lignes dans les dépendances :

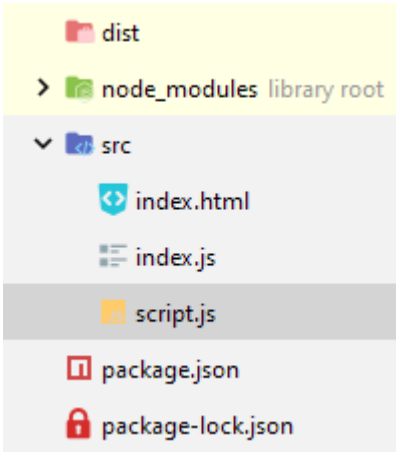
```
"devDependencies": {
  "webpack": "^5.78.0",
  "webpack-cli": "^5.0.1"
}
```

Et un dossier *node_modules* est maintenant présent dans l'architecture de notre projet.

Architecture

Pour respecter les bonnes pratiques de webpack, créer un fichier index.js (le point d'entrée défini dans le fichier *package.json*), un dossier *dist* vide et un dossier *src* qui contient nos sources (index.html et script.js).

À ce stade, l'architecture du projet est la suivante :



HtmlWebpackPlugin

Installer un plugin pour simplifier la création d'un HTML minifié dans le dossier *dist*.

```
npm install --save-dev html-webpack-plugin
```

Faire l'import de notre fichier script.js à l'intérieur du point d'entrés *index.js*.

```
import './script.js';
```

Configuration

Créer un fichier *webpack.config.js* qui contiendra toute la configuration de webpack.

Depuis Webpack 5, la création d'un fichier *webpack.config.js* est fortement recommandé

```
const path = require('path');
const HtmlWebpackPlugin = require("html-webpack-plugin");

module.exports = {
    entry: './src/index.js',
    plugins: [
        new HtmlWebpackPlugin(
            {
                template: './src/index.html'
            }
        )
    ],
    output: {
        filename: '[name].bundle.js',
        path: path.resolve(__dirname, 'dist'),
        clean: true
    }
};
```

Ce fichier précise :

- qu'index.js est le point d'entrée,
- qu'on va utiliser le plugin HtmlWebpackPlugin
- que webpack devra générer :
  - un fichier html à partir du fichier *index.html* du dossier *src*
  - un fichier javascript à partir du fichier *index.js* du dossier *src*
  - lier le javascript et le html qui ont été générés dans le dossier *dist*

```
npx webpack          // Permet la génération des fichiers dans le dossier dist
npw webpack --watch  // Utile en developpement (Hot reloading)
```

Tailwind

Cette étape est facultative. Elle permet de montrer comment installer un vrai framework scss dans un projet webpack. Bien entendu, n'importe quel framework css peut être utilisé avec un CDN.

Installer Tailwind en tant que plugin webpack post-css

```
npm install --save-dev tailwindcss postcss autoprefixer
npx tailwindcss init
```

Creer le fichier *postcss.config.js*

```
module.exports = {
    plugins: {
        tailwindcss: {},
        autoprefixer: {}
    }
}
```

Modifier le fichier *tailwind.config.js* pour lui indiquer où se trouve les fichiers html et javascript

```
module.exports = {
  content: ["./src/**/*.{html,js}"],
  theme: {
    extend: {}
  },
  plugins: []
}
```

Créer un fichier css *style.css* dans notre projet en y ajoutant les directives de tailwind :

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

Ajouter le fichier *main.css* aux imports du point d'entrée de webpack *index.js*.

Webpack a besoin de "loaders" pour pouvoir charger Tailwind.

On les installe :

```
npm install style-loader css-loader postcss-loader
```

Et on ajoute les "loaders" en tant que module dans le fichier *webpack.config.js*

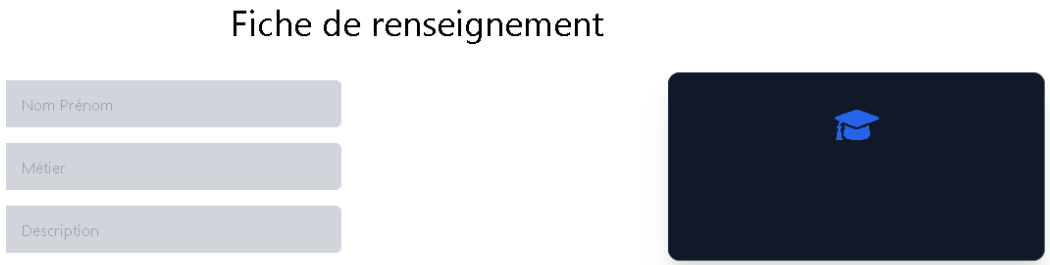
```
module: {
    rules: [
      {
        test: /\.css$/i,
        include: path.resolve(__dirname, 'src'),
        use: ['style-loader', 'css-loader', 'postcss-loader']
      }
    ]
  }
```

Réalisation

Le fichier HTML commence par un titre H1.

Sur le côté gauche, 3 inputs. Sur le côté droit, un cadre en couleur.

L'objectif est de faire en sorte que les données inscritent dans les inputs se retrouvent dans le cadre de droite.



HTML

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>M01D02</title>
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.1.1/css/all.min.css" />
</head>
<body>
<h1 id="titre" onclick="salut()">Fiche de renseignement</h1>
<div class="flex flex-row container mx-auto">
    <div id="formulaire">
        <div class="w-1/3">
            <input type="search" placeholder="Nom Prénom" id="form_nom">
            <input type="search" placeholder="Métier" id="form_metier">
            <input type="search" placeholder="Description" id="form_description">
        </div>
    </div>
    <div id="carte">
        <i class="fa-solid fa-graduation-cap text-4xl text-blue-600"></i>
        <h3 id="carte_nom"></h3>
        <p id="carte_metier">
        <p id="carte_description">
        </p>
    </div>
</div>
</body>
</html>
```

Font awesome est intégré grâce à un CDN. Le "loader" de babel ne sera vu qu'au module 02.

Javascript

```
let form_nom = document.getElementById('form_nom');
let form_metier = document.getElementById('form_metier');
let form_description = document.getElementById('form_description');
let carte_titre = document.getElementById('carte_nom');
let carte_metier = document.getElementById('carte_metier');
let carte_description = document.getElementById('carte_description');

form_nom.addEventListener('input', () => carte_titre.innerText = form_nom.value);
form_metier.addEventListener('input', () => carte_metier.innerText = form_metier.
value);
form_description.addEventListener('input', () => carte_description.innerText =
form_description.value);
```

Utiliser uniquement les addEventListener() et pas les évènements en HTML. Pour qu'une fonction soit accessible avec des evenements de type onclick() alors elle doit etre dans le scope global de webpack. window.salut = function () {console.log('salut');, par exemple. Puis

CSS

```
@tailwind base;
@tailwind components;
@tailwind utilities;

input {
    @apply m-2 w-full h-12 px-4 py-1 rounded-r-md border-gray-800 bg-gray-300 text-
black focus:outline-none;
}

#formulaire {
    @apply basis-2/3 flex justify-center;
}

#titre {
    @apply text-4xl text-center m-4 p-4;
}

#carte {
    @apply text-center basis-1/4 block rounded-xl border border-gray-800 bg-gray-900
p-4 shadow-xl sm:p-6 lg:p-8;
}
```

```
#carte_metier, #carte_description {
    @apply mt-4 text-sm text-gray-300;
}

#carte_nom {
    @apply mt-3 text-lg font-bold text-white sm:text-xl;
}
```